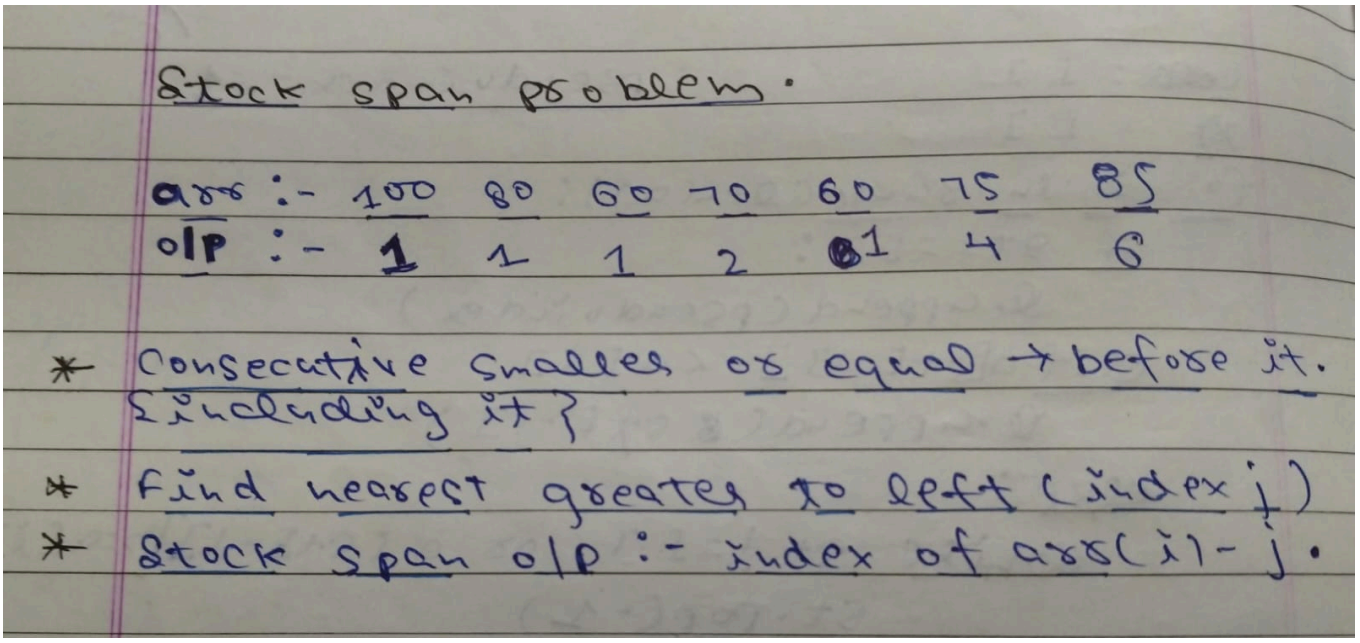
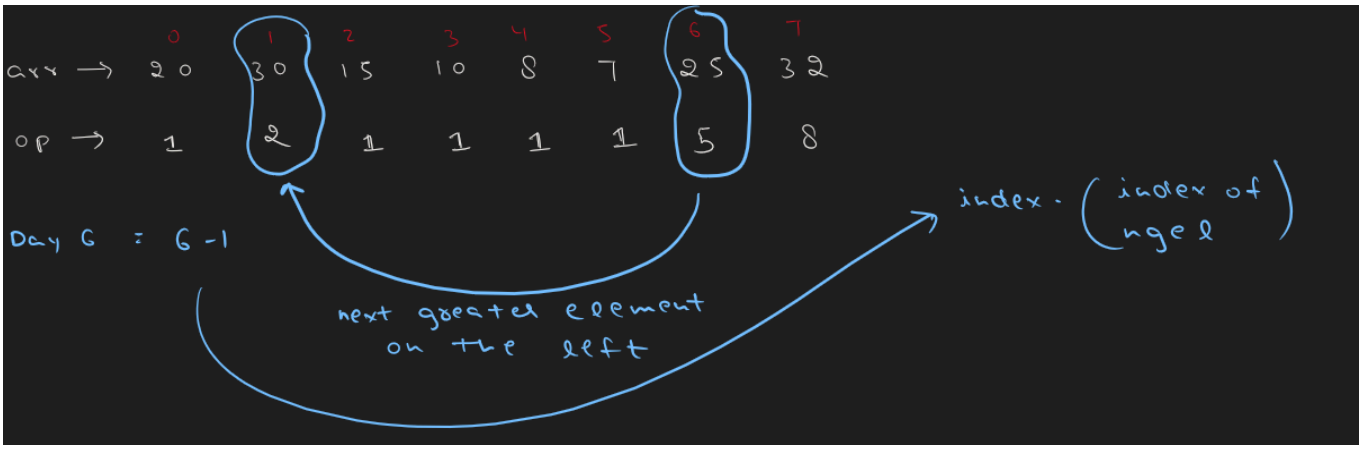
🎯 Stock Span

https://codeskiller.codingblocks.com/problems/502



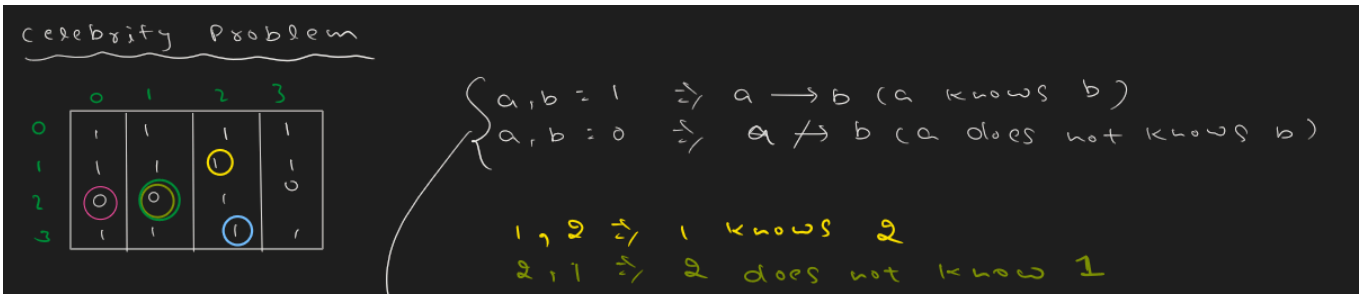


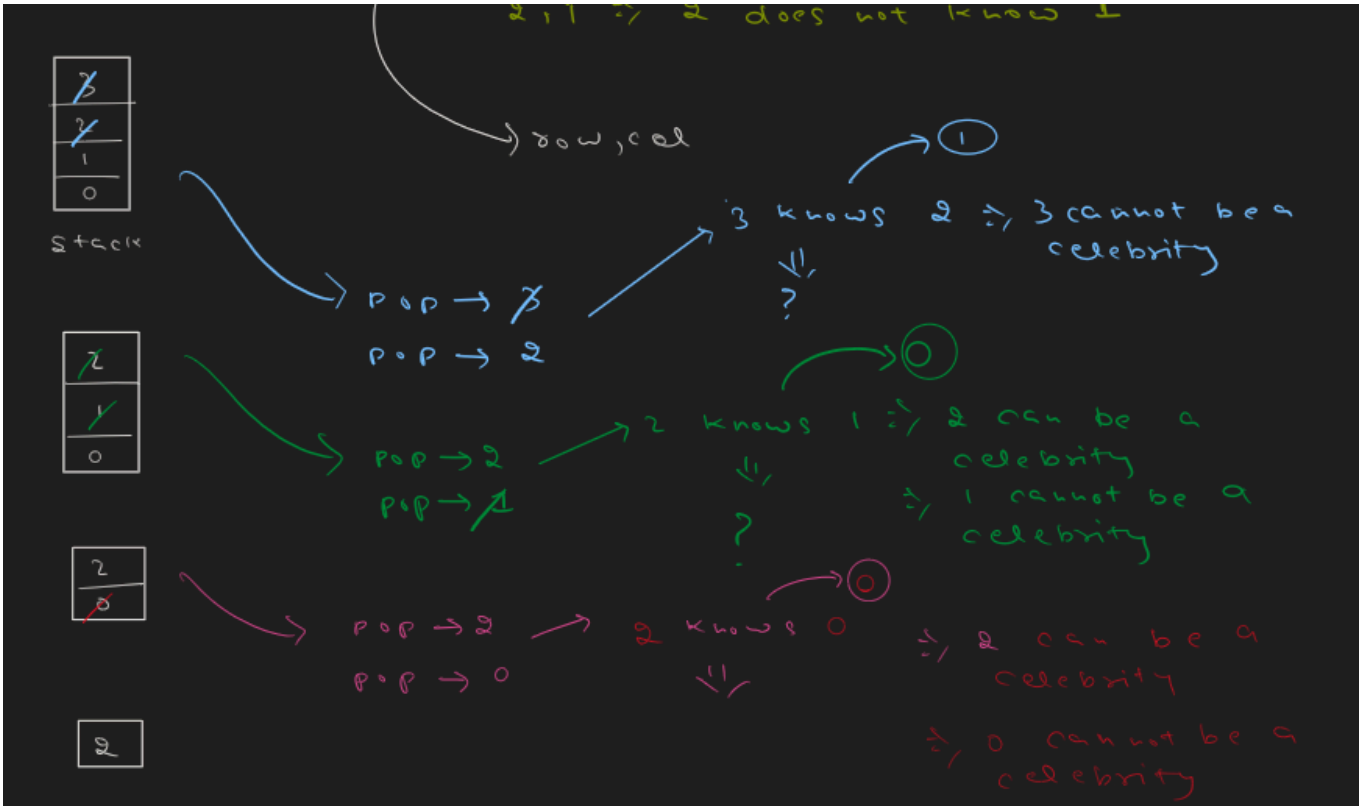

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for(int i=0; i<n; i++){
            arr[i] = sc.nextInt();
        }
        int[] ans = ngel(arr);
        // display(arr);
        display(ans);
    }


    public static int[] ngel(int[] arr) {
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];
        for (int i = 0; i<arr.length; i++) {
            if(st.isEmpty()){
                ans[i] = i-(-1);
```

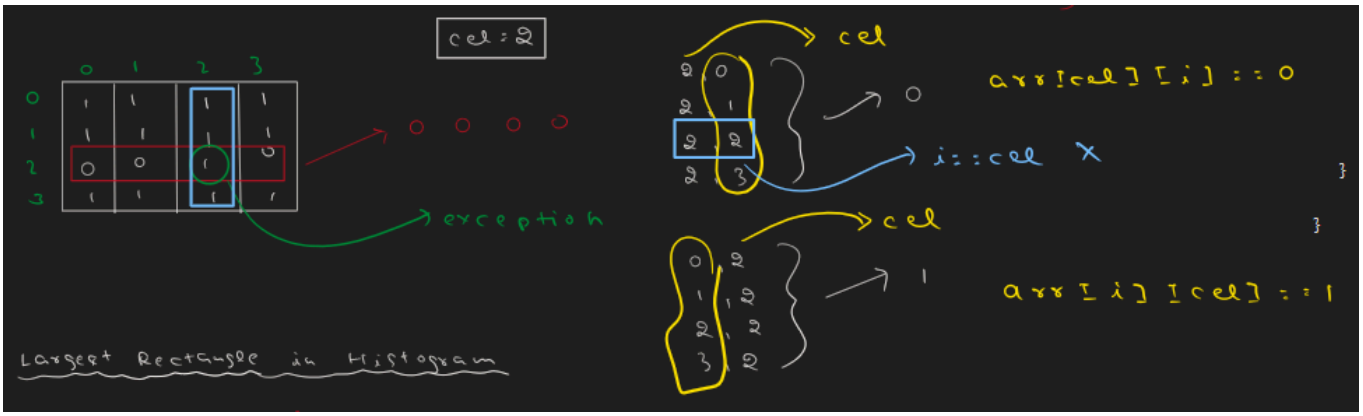
```
            }
            else if(st.peek() > arr[i]){
                ans[i] = i - st.peek();
            }
            else{
                while (!st.isEmpty() && arr[st.peek()] <= arr[i]) {
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = i-(-1);
                }
                else{
                    ans[i] = i-st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static void display(int[] a){
        for (int i = 0; i < a.length; i++) {
            System.out.print(a[i] + " ");
        }
        System.out.print("END");
    }

}
```

🎯.Find the Celebrity

https://leetcode.ca/all/277.html








```java
public class FindTheCelebrity {
      public static void main(String[] args) {
            int[][] arr = {
            { 0, 1, 1, 1 },
            { 1, 0, 1, 1 },
            { 0, 0, 0, 0 },
            { 1, 1, 1, 0 }
        };
            System.out.println(celebrity(arr));
      }

    private static int celebrity(int[][] arr) {
        Stack < Integer > st = new Stack < > ();
        for (int i = 0; i < arr.length; i++) {
            st.push(i);
        }
        while (st.size() > 1) {
            int a = st.pop();
            int b = st.pop();
            if (arr[a][b] == 1) {
                st.push(b);
            } else {
```

```
                st.push(a);
            }
        }
        int candidate = st.pop();
        for (int i = 0; i < arr.length; i++) {
            if (i == candidate) { //don't compare with self
                continue;
            }
            if (arr[candidate][i] == 1 || arr[i][candidate] == 0) {
                return -1;
            }
        }
        return candidate;
    }

}
```

🎯 Largest Rectangle in Histogram

https://leetcode.com/problems/largest-rectangle-in-histogram/

width = right[i] - left[i] - 1;

```java
public class LargestRectangleInHistogram {
    public static void main(String[] args) {
        int[] arr = {2, 1, 5, 6, 2, 3};
        System.out.println(maxArea(arr));
    }

    public static int[] nser(int[] arr) {
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];
        for (int i = arr.length - 1; i >= 0; i--) {
            if (st.isEmpty()) {
                ans[i] = arr.length;
            } else if (arr[st.peek()] < arr[i]) {
                ans[i] = st.peek();
```

```
            } else {
                while (!st.isEmpty() && arr[st.peek()] >= arr[i]) {
                    st.pop();
                }
                if (st.isEmpty()) {
                    ans[i] = arr.length;
                } else {
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int[] nsel(int[] arr) {
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];
        for (int i = 0; i < arr.length; i++) {
            if (st.isEmpty()) {
                ans[i] = -1;
            } else if (arr[st.peek()] < arr[i]) {
                ans[i] = st.peek();
            } else {
                while (!st.isEmpty() && arr[st.peek()] >= arr[i]) {
                    st.pop();
                }
                if (st.isEmpty()) {
                    ans[i] = -1;
                } else {
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int maxArea(int[] arr) {
        int[] right = nser(arr);
        int[] left = nsel(arr);
        int max = 0;
        for (int i = 0; i < arr.length; i++) {
            int width = right[i] - left[i] - 1;
            int area = arr[i] * width;
            max = Math.max(max, area);
        }
        return max;
    }
}
```

```
class Solution {
    public int largestRectangleArea(int[] heights) {
```

```
        return maxArea(heights);
    }
    public static int[] nser(int[] arr) {
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];
        for (int i = arr.length - 1; i >= 0; i--) {
            if (st.isEmpty()) {
                ans[i] = arr.length;
            } else if (arr[st.peek()] < arr[i]) {
                ans[i] = st.peek();
            } else {
                while (!st.isEmpty() && arr[st.peek()] >= arr[i]) {
                    st.pop();
                }
                if (st.isEmpty()) {
                    ans[i] = arr.length;
                } else {
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int[] nsel(int[] arr) {
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];
        for (int i = 0; i < arr.length; i++) {
            if (st.isEmpty()) {
                ans[i] = -1;
            } else if (arr[st.peek()] < arr[i]) {
                ans[i] = st.peek();
            } else {
                while (!st.isEmpty() && arr[st.peek()] >= arr[i]) {
                    st.pop();
                }
                if (st.isEmpty()) {
                    ans[i] = -1;
                } else {
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int maxArea(int[] arr) {
        int[] right = nser(arr);
        int[] left = nsel(arr);
        int max = 0;
        for (int i = 0; i < arr.length; i++) {
            int width = right[i] - left[i] - 1;
```

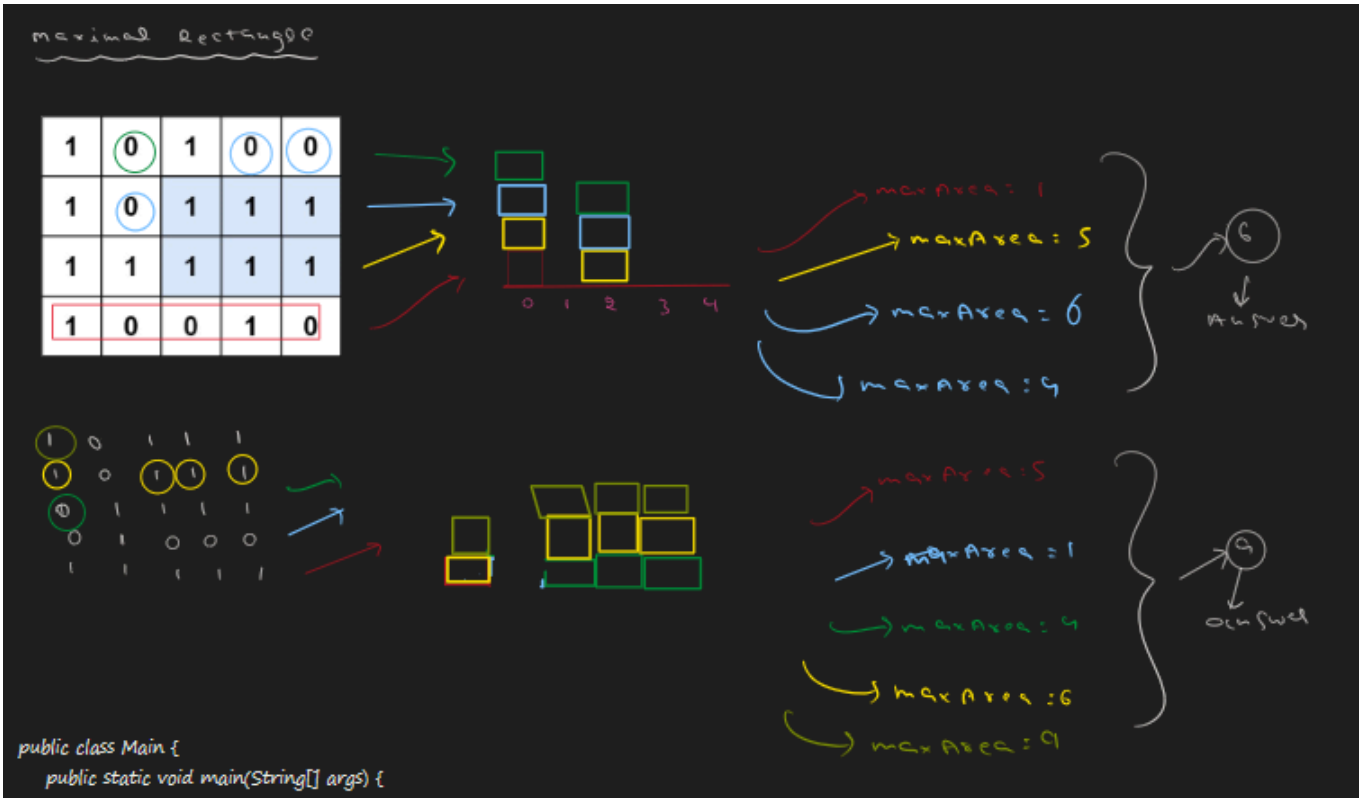
```
            int area = arr[i] * width;
            max = Math.max(max, area);
        }
        return max;
    }
}
```

🎯 Maximal Rectangle

https://leetcode.com/problems/maximal-rectangle/

💡 Intuition

As we move row by row, each column's height increases when we see a 1 and resets to zero when we see a 0. This way, every row represents a histogram of consecutive 1s up to that point. For each such histogram, we find the largest rectangle area, and the maximum among all gives the final answer.




```
public class MaximalRectangle {
      public static void main(String[] args) {
            char[][] matrix = {
            { '1', '0', '1', '0', '0' },
            { '1', '0', '1', '1', '1' },
            { '1', '1', '1', '1', '1' },
            { '1', '0', '0', '1', '0' }
        };

        int[] arr = new int[matrix[0].length];
        int ans = 0;
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (matrix[i][j] == '1') {
                    arr[j]++;
                } else {
```

```
                arr[j] = 0;
            }
        }
        ans = Math.max(ans, maxArea(arr));
    }
    System.out.println(ans);
}

public static int[] nser(int[] arr) {
    Stack < Integer > st = new Stack < > ();
    int[] ans = new int[arr.length];
    for (int i = arr.length - 1; i >= 0; i--) {
        if (st.isEmpty()) {
            ans[i] = arr.length;
        } else if (arr[st.peek()] < arr[i]) {
            ans[i] = st.peek();
        } else {
            while (!st.isEmpty() && arr[st.peek()] >= arr[i]) {
                st.pop();
            }
            if (st.isEmpty()) {
                ans[i] = arr.length;
            } else {
                ans[i] = st.peek();
            }
        }
        st.push(i);
    }
    return ans;
}

public static int[] nsel(int[] arr) {
    Stack < Integer > st = new Stack < > ();
    int[] ans = new int[arr.length];
    for (int i = 0; i < arr.length; i++) {
        if (st.isEmpty()) {
            ans[i] = -1;
        } else if (arr[st.peek()] < arr[i]) {
            ans[i] = st.peek();
        } else {
            while (!st.isEmpty() && arr[st.peek()] >= arr[i]) {
                st.pop();
            }
            if (st.isEmpty()) {
                ans[i] = -1;
            } else {
                ans[i] = st.peek();
            }
        }
        st.push(i);
    }
    return ans;
}
```

```
    public static int maxArea(int[] arr) {
        int[] right = nser(arr);
        int[] left = nsel(arr);
        int max = 0;
        for (int i = 0; i < arr.length; i++) {
            int width = right[i] - left[i] - 1;
            int area = arr[i] * width;
            max = Math.max(max, area);
        }
        return max;
    }

}
```

```
class Solution {
    public int maximalRectangle(char[][] matrix) {
        int[] arr = new int[matrix[0].length];
        int ans = 0;
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                if (matrix[i][j] == '1') {
                    arr[j]++;
                } else {
                    arr[j] = 0;
                }
            }
            ans = Math.max(ans, maxArea(arr));
        }
        return ans;
    }

    public static int[] nser(int[] arr) {
        Stack < Integer > st = new Stack < > ();
        int[] ans = new int[arr.length];
        for (int i = arr.length - 1; i >= 0; i--) {
            if (st.isEmpty()) {
                ans[i] = arr.length;
            } else if (arr[st.peek()] < arr[i]) {
                ans[i] = st.peek();
            } else {
                while (!st.isEmpty() && arr[st.peek()] >= arr[i]) {
                    st.pop();
                }
                if (st.isEmpty()) {
                    ans[i] = arr.length;
                } else {
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int[] nsel(int[] arr) {
        Stack < Integer > st = new Stack < > ();
        int[] ans = new int[arr.length];
```

```
        for (int i = 0; i < arr.length; i++) {
            if (st.isEmpty()) {
                ans[i] = -1;
            } else if (arr[st.peek()] < arr[i]) {
                ans[i] = st.peek();
            } else {
                while (!st.isEmpty() && arr[st.peek()] >= arr[i]) {
                    st.pop();
                }
                if (st.isEmpty()) {
                    ans[i] = -1;
                } else {
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int maxArea(int[] arr) {
        int[] right = nser(arr);
        int[] left = nsel(arr);
        int max = 0;
        for (int i = 0; i < arr.length; i++) {
            int width = right[i] - left[i] - 1;
            int area = arr[i] * width;
            max = Math.max(max, area);
        }
        return max;
    }

}
```

🎯 Valid Parentheses

https://leetcode.com/problems/valid-parentheses/

```
class Solution {
    public boolean isValid(String str) {
        Stack < Character > st = new Stack < Character > ();
        for (int i = 0; i < str.length(); i++) {
            char ch = str.charAt(i);
            if (ch == '(' || ch == '[' || ch == '{')
                st.push(ch);
            else if (ch == ')' || ch == ']' || ch == '}') {
                if (st.empty())
                    return false;
                int alpha = st.pop();
                if (alpha == '(')
                    if (ch != ')')
                        return false;
                if (alpha == '[')
                    if (ch != ']')
                        return false;
                if (alpha == '{')
```

```
                    if (ch != '}')
                        return false;
                }
            }
            if (st.empty())
                return true;
            else
                return false;
        }

}
```

🎯 Baseball Game

https://leetcode.com/problems/baseball-game/

```
class Solution {
    public int calPoints(String[] ops) {
        Stack<Integer> stack = new Stack<>();
        int sum = 0;

        for (String op : ops) {
            if (op.equals("C")) {
                int removed = stack.pop();
                sum -= removed;
            } else if (op.equals("D")) {
                int doubled = 2 * stack.peek();
                stack.push(doubled);
                sum += doubled;
            } else if (op.equals("+")) {
                int top = stack.pop();
                int second = stack.peek();
                int current = top + second;
                stack.push(top);
                stack.push(current);
                sum += current;
            } else {
                int score = Integer.parseInt(op);
                stack.push(score);
                sum += score;
            }
        }
        return sum;
    }

}
```

🔹 What is a Doubly Ended Queue?

In a normal queue, elements can be **added from one end** (the rear) and **removed from the other end** (the front).

In a **doubly ended queue (deque)**, elements can be **added and removed from both ends**.
That means both the **enqueue (add)** and **dequeue (remove)** operations can be performed from either side.